Analyzing Real Cluster Data for Formulating Allocation Algorithms in Cloud Platforms

Olivier Beaumont, Lionel Eyraud-Dubois, Juan-Angel Lorenzo-Del-Castillo

Analyzing Real Cluster Data for Formulating Allocation Algorithms in Cloud Platforms

Olivier Beaumont, Lionel Eyraud-Dubois, Juan-Angel Lorenzo-del-Castillo

Inria Bordeaux Sud-Ouest

Université de Bordeaux

Abstract

A problem commonly faced in Computer Science research is the lack of real usage data that can be used for the validation of algorithms. This situation is particularly true and crucial in Cloud Computing. The privacy of data managed by commercial Cloud infrastructures, together with their massive scale, makes them very uncommon to be available to the research community. Due to their scale, when designing resource allocation algorithms for Cloud infrastructures, many assumptions must be made in order to make the problem tractable.

This paper provides deep analysis of a cluster data trace recently released by Google and focuses on a number of questions which have not been addressed in previous studies. In particular, we describe the characteristics of job resource usage in terms of dynamics (how it varies with time), of correlation between jobs (identify daily and/or weekly patterns), and correlation inside jobs between the different resources (dependence of memory usage on CPU usage). From this analysis, we propose a way to formalize the allocation problem on such platforms, which encompasses most job features from the trace with a small set of parameters.

Keywords: Cloud Computing, Data Analysis, Parallel jobs

1. Introduction

The topic of resource allocation algorithms for Cloud Computing platforms has been the focus of recent interest. In this context, the objective is to allocate a set of services –understanding a service as a divisible workload– onto a set of physical machines, so as to optimize resource usage, or to ensure strong Quality of Service guarantees, or to limit the number of migrations used, among others. On the algorithmic side, several solutions and techniques have been proposed to

Email addresses: olivier.beaumont@inria.fr (Olivier Beaumont), lionel.eyraud-dubois@inria.fr (Lionel Eyraud-Dubois), juan-angel.lorenzo-del-castillo@inria.fr (Juan-Angel Lorenzo-del-Castillo)

compute such allocations: many are based on variants of standard Bin Packing algorithms (like First Fit or Best Fit), some are more focused on the dynamic aspect and provide online rules to compute new valid allocations. There is no clear consensus in the community on which aspects of the problem are the most important (dynamicity, fault tolerance, multidimensional resources, additional user-supplied constraints, ...), neither on the formal algorithmic models to take such aspects into account.

In this paper, we analyze a cluster usage trace recently released by Google with the goal of providing answers to some of these questions. Some of the characteristics of this trace have already been analyzed in [1]. Our objective in this paper is twofold. Firstly, we aim at finding new characteristics of the trace and to exhibit the main properties of the jobs running on it. Second, we aim at proposing a set of very few parameters that still account for the main characteristics of the trace and may leverage both the generation of realistic random traces and the design of efficient allocation algorithms.

1.1. The Google Cluster Trace

Google recently released a complete usage trace from one of its production clusters [2]. The workload consists in a massive number of *services* (called *jobs* in the terminology of this trace), which can be further divided into *tasks*, being each task assigned to a single physical machine. Note that a task is an indivisible piece of work that can be implemented as a program running in a virtual machine (VM) or –as it actually happens in the trace– in a Linux Container (LxC). The data are collected from 12583 heterogeneous machines, span a time of 29 days and provide exhaustive profiling information –such as the memory and CPU usage of each task– on 5-minute monitoring intervals (called *time windows* in the rest of this paper). Each job has a priority assigned, 0 being the lowest one and 11 the highest one. According to [3, 4], priorities can be grouped into *infrastructure* (11), *monitoring* (10), *normal production* (9), *other* (2-8) and *gratis (free)* (0-1). The scheduler generally gives preference to resource demands from higher priority tasks over tasks belonging to lower priority groups, to the point of evicting the latter ones if needed. All data, as well as the nature of the workloads, have been obfuscated. The trace providers state that the data come mainly from "internal workloads" and mention MapReduce as an example of a program that spawns two jobs (a master and a slave), each comprising multiple tasks. Let us note that the goal of this paper is not to understand the behavior of the scheduling and resource allocation algorithm used by Google to allocate tasks and jobs, but rather to concentrate on the general properties of the jobs themselves and how to describe the trace.

The whole trace is split into numerous files with a total size of 186 Gb. These trace files contain thorough information about the platform and the jobs running on it. In practice, one of the main difficulty is related to the size of the database and the time needed to validate any assumption based on these data. In this paper, we propose an extraction of the database containing all information about the jobs that we consider as dominant, *i.e.* jobs that at some

instant belong to the minimal set that still accounts for most of the resources usage, and that can be used to test and validate more easily new assumptions.

1.2. Relevant Features and Important Questions

Based on the trace of the production clusters described in [2], we will in particular concentrate on the following questions :

Static: It has been noticed in [1] that, at a given time step, a small portion of jobs only accounts for most of the usage of the platform (both in terms of CPU usage, memory consumption, number of assigned tasks,...). In this study, we confirm this observation. In particular, we prove that less than 6% of all jobs account for almost 95% of CPU usage and 90% of memory usage. A crucial consequence is that, in this context, it is possible to design efficient and optimized algorithms to allocate these jobs, contrarily to what is often assumed in the literature, while possibly relying on basic on-line linear complexity heuristics to schedule the huge number of very small remaining jobs. We also provide a detailed analysis of the class of priorities of these jobs and of the number of tasks they involve.

Dynamics: In the context of the design of efficient algorithms, the dynamics of jobs have also to be considered. Indeed, as already stated, we observe statically (at any time stamp) that the number of dominant jobs necessary to account for most of the platform usage is relatively small. Related to the dynamics of the system, we will consider the following additional questions: does the set of dominant jobs vary over time or is it relatively stable? What is the distribution of lifespans of dominant jobs? How much of the overall CPU usage does the set of stable dominant jobs account for? Does the usage (memory, CPU, tasks) of dominant stable jobs vary over time and, if it is the case, do they exhibit specific (hourly, daily, weekly) patterns? In the context of resource allocation mechanisms, this information is crucial in order to determine, even for the small set of dominant jobs, what can be done statically for a long period and what needs to be recomputed at each time step.

Advanced features of the jobs: As stated above, a priori, each job comes with a given number of tasks (or with an overall demand in the case of web services for instance) in terms of CPU, memory, disk, bandwidth... This leads to multi-dimensional packing problems that are notoriously difficult to solve, even for relatively small instances (a few hundreds of items). Nevertheless, multi-dimensional packing problem are greatly simplified if there are correlations between the different dimensions. In this paper, we will mostly concentrate on CPU-Memory correlation. We will prove (by depicting for all the tasks from dominant jobs, at all time stamps, their memory and CPU footprints) that for a large fraction of the jobs, the CPU usage varies with time and from task to task but that the memory usage is (almost) independent of the CPU usage. More precisely, we will classify the jobs into a few categories.

Fault Tolerance Issues: In such a large platform (typically involving more than 10k nodes), failures are likely to happen. In the case when a SLA has been established between the provider and the client, replication or checkpointing strategies have to be designed in order to cope with failures. In this paper,

we will concentrate on the following questions: what is the actual frequency of failures on such large scale production cluster? Are failures heavily correlated (a relatively large number of nodes fail simultaneously) or mostly independent? Again, these observations can be later used in order to validate failure models or to precisely quantify the qualities and limits of a model.

The rest of this paper is organized as follows. Section 2 presents the related work and clarifies our contribution. In Section 3, we define the set of *dominant* jobs which account for most of the resource usage and provide a simple description. In Section 4, we analyze the static properties of the workload of this dominant set of jobs. More dynamic properties following the evolution of jobs over time are considered in Section 5. Issues related to failure characterization will be considered in Section 6. Section 7 proposes a variety of reasonable simplifying assumptions which can be made when designing allocation algorithms. Concluding remarks are presented in Section 9. Note that it is impossible to include in this paper all the plots and numerical results and we will present throughout the text a small set of examples spanning all the questions above.

2. Related work

The trace has been published by Google in [2] and its format described in [3]. A forum to discuss trace-related topics is available in [5]. Several teams have analyzed this trace and the associated workload. Reiss et al. made a general analysis in [1] and [4], paying special attention to the workload heterogeneity and variability of the trace and confirming the need for new cloud resource schedulers and useful guidance for their design. Di et al. [6] used K-means clustering techniques in an attempt to classify the type of applications running in the trace based on their resource utilization. As a previous step, a comparison of these applications with respect to those typically run on Grids was done in [7]. In [8], Liu and Cho focused on frequency and pattern of machine maintenance events, as well as basic job-and task-level workload behavior. Garraghan et al. made in [9] a coarse-grain statistical analysis of submission rates, server classification, and wasted resource server utilization due to task failure.

Other teams have used the trace as an input for their simulations or predictions. Di et al. used it in [10] to test fault-tolerance optimization techniques based on a checkpointing/restart mechanism, and in [11] to predict host load using a Bayesian model. Amoretti et al. [12] carried out an experimental evaluation of an application-oriented QoS model. On a different context, job burstiness is used as an input for a queue-based model of an elasticity controller in [13].

All these approaches have been carried out either without any particular application in mind or just as an input to test simulations. On the contrary, our analysis aims at discriminating those features from the trace that are relevant to define a model which captures the main characteristics of both jobs and machines, that is tractable from an algorithmic point of view (not too many parameters) and that can later be used for random generation of realistic traces.

On the other hand, some work has already been done to analyze the dynamics of resource usage in datacenters. Bobroff et al [14] proposed a dynamic consolidation approach to allocate VMs into physical servers reducing the amount needed to manage SLA violations. They analyze the historical usage data and create an estimated model, based on Auto Regressive processes, to forecast the probability function of future demand. Gmach et al [15] rely on pattern recognition methods to generate synthetic workloads to represent trends of future behavior. In this paper, we analyze a much larger dataset, which is publicly available, and we analyze both the dynamics and the repartition of resource usage to provide a set of well-justified assumptions for the design of allocation algorithms.

3. Dominant jobs

In this section, we analyze the repartition of resource usage between jobs, and more precisely of those jobs which are crucial when designing efficient allocation algorithms since they account for most of platform usage. Following [1], we will prove that at any time step, a small portion of the jobs accounts for a large portion of CPU and memory usage. Therefore, reasonable resource allocation algorithms should concentrate on a careful allocation of these *dominant jobs*.

As mentioned before, the trace contains monitoring data averaged over 5-minute time windows. From each window to the following one, the number of running jobs will vary to a larger or lower extent, depending on the dynamics of the trace in that period. Hence, jobs that account for most of resource usage at a time window will not necessarily do it in the next one. We looked for a set of jobs that were representative of the whole trace. We define D_i, the set of dominant jobs for a given time window i, as the smallest set of jobs which account for at least 90% of the CPU usage of this time window. The set of *dominant jobs* which we consider in this paper is the union of these sets over the whole trace, $D = \bigcup_i D_i$. This set D contains 25839 jobs, which makes 3.8% of all jobs in the trace.

Table 1 shows the number of alive jobs in D and its resource usage $R(D)$ (CPU and memory) relative to the corresponding values on each time window

Stats	*Ratios*			
	$\|D\|$	$\|D\|/Alljobs_{tw}$	R_{cpu}	R_{mem}
mean	240.9	0.058	0.947	0.890
std	13.8	0.003	0.009	0.008
min	203	0.050	0.905	0.840
25%	232	0.056	0.942	0.886
50%	239	0.058	0.947	0.891
75%	248	0.059	0.953	0.895
max	336	0.082	0.975	0.913

Table 1: Number of dominant jobs, and ratios of number of jobs, cpu and memory usage w.r.t. all other jobs, per time window.

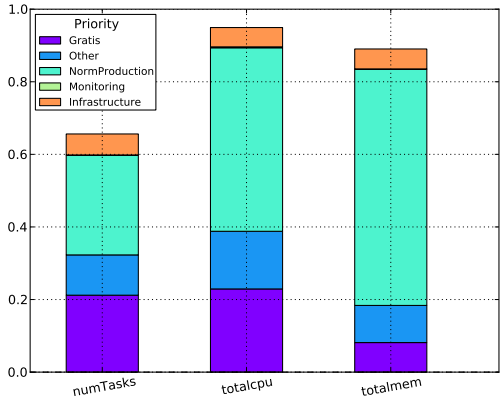


Figure 1: Average resource usage of dominant jobs stacked by priority class.

(tw). The first row shows that, in average, 240.9 jobs (*i.e.* 5.8% of the jobs running on each tw) account for 94.7% of the overall CPU usage and 89% of the memory usage. This confirms that, even though the overall trace involves a huge number of jobs, it is of interest to concentrate on a very small fraction of them that accounts for most of platform usage at any time. In the rest of the paper we will focus on these dominant jobs only.

The small amount of jobs that comprises the dominant set is a crucial observation when considering the design of resource allocation algorithms. Indeed a quadratic (resp. cubic) complexity resource allocation would be 1000 (resp. $4 \cdot 10^4$) times faster on this small set of jobs rather than on the whole set, while still optimizing the allocation of the jobs that account for almost all of the CPU usage. It can be noted that these jobs involve a large number of tasks (470 on average, for 49983 the largest one), what explains their overall weight. It is therefore crucial for the algorithms to concentrate on jobs and not on tasks (or virtual machines).

A second observation that can be made is that, for those jobs that account for 90% of the CPU usage, most of the workload is dominated by jobs from particular priority groups. Figure 1 shows three parameters of the trace (number of tasks, total CPU and memory usage) normalized to the total cluster utilization and stacked by priority classes. From this, it can be observed that the set of dominant jobs, which accounts for more than 94% of CPU usage and almost 90% of the overall memory usage, comprises only 65% of the overall number of tasks. The memory and CPU usage bars show that most of jobs belong to the *Normal Production* priority class. Jobs in that priority class are prevented from being evicted by the cluster scheduler in case of over-allocation of machine resources. The second most important set of jobs are the ones in the *Gratis* priority class. *Gratis* priority class jobs are defined as those that incur little

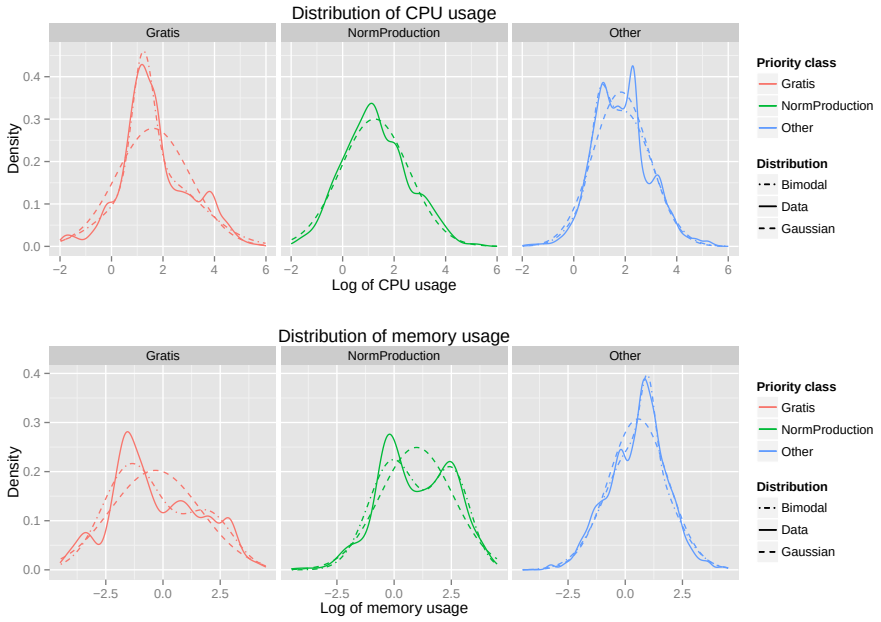


Figure 2: Usage density distributions of dominant jobs for CPU (top) and memory (bottom).

internal charging, and therefore can be easily evicted by the cluster scheduler if needed due to over-allocation. Note that, according to the leftmost bar from the figure, the number of tasks from jobs in the *Gratis* priority class is almost as large as those of *Normal Production* priority (21% versus 27%, respectively).

4. Workload characterization

In this section, we analyze more precisely the static behavior of the workload of dominant jobs. We first describe the distribution of total resource usage for jobs in this set, and find that it can be summarized by a log-normal or a mixture of two log-normal distributions. Then, we analyze the relationships between memory and CPU usage of individual tasks inside a job. Indeed, in the most general setting, resource allocation is amenable to a multi-dimensional bin packing problem (where the dimensions correspond to CPU, memory, disk,...), that are known to be NP-Complete and hard to approximate (see for instance [16] for the offline case and and [17] for the online case). On the other hand, being able to find simple models relating CPU and memory usage can dramatically reduce the complexity of packing algorithms.

4.1. Job resource usage

For this analysis, we have computed the average CPU and memory usage of all jobs in the dominant set, and we have analyzed the behavior of the logarithm

Classes	*Gaussian*		*Gaussian mixture*					
	μ	σ	λ_1	μ_1	σ_1	λ_2	μ_2	σ_2
CPU								
Production	1.24	1.33						
Gratis	1.62	1.43	0.68	1.79	1.68	0.32	1.23	0.41
Other	1.84	1.10	0.11	1.02	0.30	0.89	1.94	1.12
Memory								
Production	0.98	1.60	0.60	-0.04	1.08	0.40	2.54	0.81
Gratis	-0.32	1.97	0.69	-1.36	1.28	0.31	1.98	1.08
Other	0.57	1.30	0.88	0.51	1.37	0.12	0.99	0.31

Table 2: Parameters of distributions used in Figure 2. For the mixture distributions, λ_1 and λ_2 are the weights associated to the two normal distributions with given μ and σ.

of these values. The result is shown on Figure 2, and shows that these distributions can be roughly approximated by a Gaussian distribution, what implies that the actual resource usage can be approximated by a log-normal distribution. However, for more precision, it is possible to model them using a mixture of two log-normal distributions. The parameters of the distributions used are given in Table 2. It is of interest to note that although *Normal Production* jobs account for most of the resource usage in total, individual jobs in the other priority classes (*Gratis* and *Other*) use more resource but, since their duration is shorter, fewer of them are alive at any given time.

4.2. CPU vs Memory Usage

In this section, we will concentrate on memory and CPU dimensions, for which data is available in the trace. Other dimensions (bandwidth, for example) are not provided. In fact, the trace provider strongly advises not to speculate about them.

Let us consider the case of a job corresponding to a web service typically handling requests. The memory footprint comes from two different origins. There is first a constant memory footprint that corresponds to the static size of the code being executed. Then, there is a memory footprint that is due to dynamic memory allocations and that is typically expected to be proportional to the number of requests being handled by the task, and therefore to its CPU usage.

In order to find out if there exists any correlation between memory and CPU, we sampled the resource usage of the set of dominant jobs at 20 random timestamps per day, which makes 600 samples along the trace. Note that some of these jobs might never run simultaneously, but we are interested here in the characteristics of each individual job. For each job, we have analyzed simultaneously the memory and CPU usage of each of their individual tasks at all timestamps when this job is alive.

Observations show that for some jobs, there exists groups with different memory-CPU utilization. So, before the analysis, patterns were clustered into

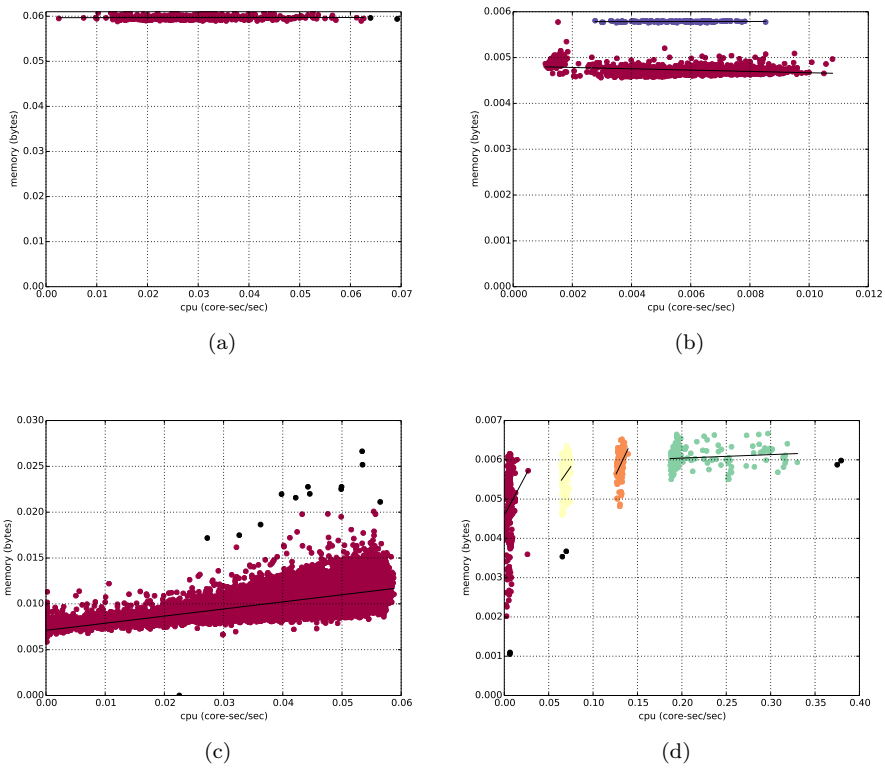


Figure 3: Dependency of memory usage vs CPU consumption of four different jobs. Colors on each group of dots show a cluster discovered by the DBSCAN algorithm.

groups using a DBSCAN algorithm. DBSCAN uses as a metric distances between nearest points and, therefore, views clusters as areas of high density separated by areas of low density [18]. After this clustering, we performed simple linear regression on each discovered cluster. We used the *r-squared* value from the linear regression and the *coefficient of variation* (standard deviation of each job's distance to the regression line divided by the mean value of all dots) to decide whether the linear regression was a good approximation for this cluster behavior, and we conservatively rejected as *BadlyFitted* all jobs in which at least one cluster was not approximated correctly. This selection rejected about half of the considered jobs. For the other half, patterns were classified into *Flat* or *Slope* by using a threshold on the *range* parameter, defined as the ratio between the value estimated by the linear regression for the median CPU usage and for 0 CPU usage.

In Figure 3, each plot corresponds to a job and each dot (*cpu,memory*) shows the specific usage profile of each job's task. Each color is associated to a specific cluster. Typical situations are depicted on each plot. The top left figure

corresponds to the case where the memory usage is independent of the CPU usage. According to the classification algorithm, this situation occurs in **67%** of all the fitted cases. A comparable situation is depicted in top right figure. In this case, the job comprises two different clusters, and could be considered for resource allocation issues as two different jobs. According to the classification, this situation accounts for **6.5%** of the fitted jobs. Therefore, in **72.5%** of the cases, it is possible to assume that the memory footprint of a job's task is independent of their CPU usage. The situation depicted in the bottom left figure corresponds to the case where the memory footprint is affine in the CPU usage, with a non-flat slope (corresponding to the previous case depicted by top figures). According to our classification, this occurs in **26.5%** of the cases. There also exists a certain amount of jobs that does not follow any clear pattern, and which were classified as *BadlyFitted*. An example is depicted on the bottom right figure. In this case, it seems that the job consists in many sub-jobs, all tasks in each sub-job being assigned the exact same CPU but varying memory.

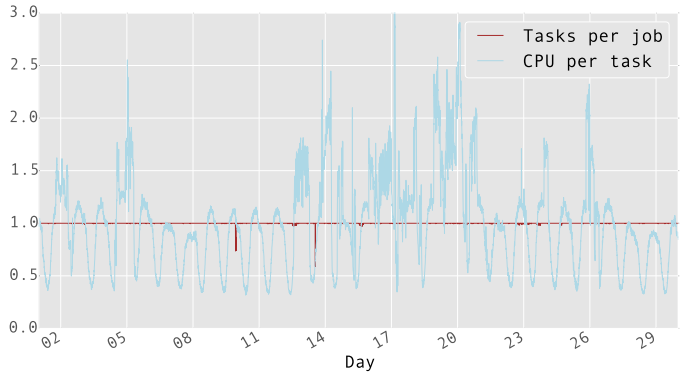
Nevertheless, we can observe that for an important fraction of jobs, and with a very conservative selection, it is possible to model memory usage as an affine function of the CPU usage (thus involving only 2 parameters per job) and that for most of these jobs, it is even possible to model the memory usage as a constant (1 parameter per job). As proven in [19], this assumption strongly simplifies resource allocation problems in Clouds.

4.3. Task distribution

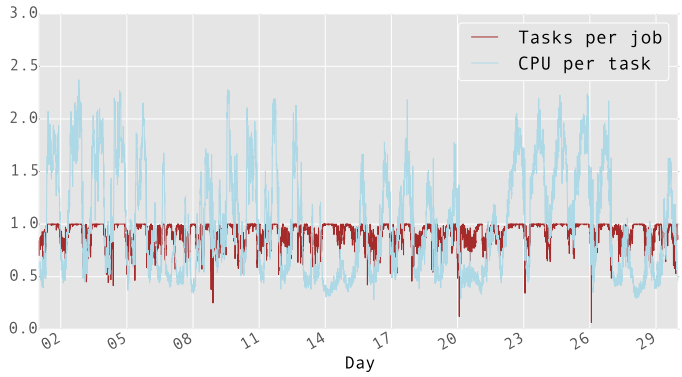
Achieving an efficient resource allocation implies considering the amount of work to be distributed among the available machines and how it can be balanced. In the Google trace, each job first requests an amount of required resources, and then its workload is spread among a number of tasks. We asserted earlier in this paper that allocation strategies must focus on jobs and not on tasks. In fact, if the resource usage of all tasks for each job grows and shrinks uniformly, then the effort must be made on allocating the amount of resources required by each job onto the available machines. To confirm our assertion, we looked at the task distribution of dominant jobs and posed several interrelated questions: *What is the variation rate of the number of tasks per job? Is it correlated to their resource usage? Is the resource usage balanced among tasks? How and how much do their usage patterns differ among jobs and over time?*

To answer these questions, we first observed the number of tasks per job. For each job, we calculated the interdecile ratio of the number of tasks at all time windows to estimate the task dispersion over time. We found out that **50%** of dominant jobs have an interdecile ratio of 1 (hence their number of tasks does not vary during the job execution) and **67%** have an interdecile ratio equal or lower than 2. So, by and large, the number of tasks remains approximately constant while the CPU usage per task varies over time. Figure 4a illustrates this case with a paradigmatic example of a typical job. Figure 4b, however, portrays an example of a less frequent case in which both the number of tasks and the resource usage per task have a strong variation. In the example with a constant number of tasks, the average amount of CPU used by each task is

clearly different at every time window. In the second example, we found that both the number of tasks and the CPU by task vary with time. According to the information from the Google forum, the scheduler does not have such a granularity, so it is likely that the variation in the number of tasks and the load of each one is rather due to the job itself.



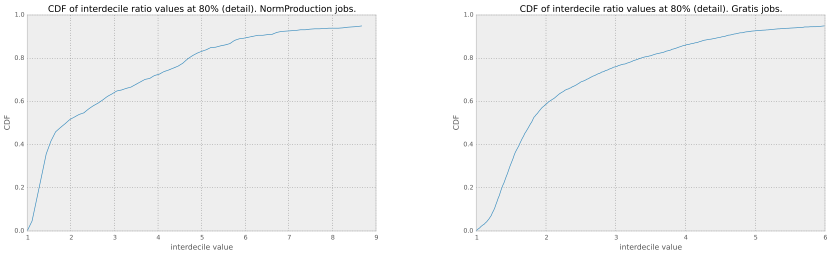
(a) Job with a constant number of tasks over time.



(b) Job with a variable number of tasks over time.

Figure 4: Normalized number of tasks and CPU usage per task for two example jobs.

A way to estimate the balance and variation in the CPU usage among tasks of a same job consists in looking at the usage distribution at each time window. A narrow distribution (quantified as a small interdecile ratio) will suggest a load well balanced among tasks. To estimate the stability of this load balance over time, we computed the cumulative density function (CDF) of the interdecile ratios at all time windows and looked at which percentage of those remained



(a) Normal Production jobs. (b) Gratis jobs.

Figure 5: Distribution of 80% threshold for interdecile ratios. A point at $(2, 0.5)$ means that for 50% of jobs, in 80% of their active time windows, the interdecile ratio is below 2.

small. For those jobs with a constant number of tasks, occurrences of high values in the CDF will suggest that the load becomes unevenly distributed at certain periods of the job's life.

After getting the CDF of each job, we aggregated the variation rates of all dominant jobs by calculating the percentage of jobs for which 80% of their interdecile ratios were under a given threshold. We found out that **52%** (resp. **59%**) of jobs have 80% of their interdecile ratios below 2 for the *Normal Production* (resp. *Gratis*) priority group. The complete distribution of these 80% threshold interdecile ratios is shown on Figure 5. These results make reasonable to conclude that the cluster workload can be modeled as a set of jobs with a stable number of tasks which show a moderate variation in their load balance, and leverage the design of algorithmic models as those proposed in Section 8.

5. Dynamic features

5.1. Dynamics of Dominant Jobs

In this section, we consider the dynamics of the dominant jobs identified earlier. The first question to consider is about the "stability" of this set of dominant jobs: how much does the set of dominant jobs change over time? To provide answers to this question, we have analyzed the distribution of their durations, which is shown on Table 3 for the three main priority classes. We can see that the *Gratis* and *Other* priority classes have similar behavior: most jobs in these classes last for a very short time (half of them last less than 25 minutes), but some last much longer (1% of *Gratis* jobs last more than 30 hours, 1% of *Other* jobs last more than 15 hours). However, the *Normal Production* jobs are much more stable: half of them run for more than 31.7 hours. In fact, about 15.6% of all *Normal Production* dominant jobs run for the whole trace.

Another interesting question arises when observing the variation of resource usage of jobs: we have noticed that the CPU usage of *Normal Production* jobs follows interesting periodic patterns. In the following subsection, we analyze how much correlation exists between the CPU usage of individual jobs.

Stats	*Duration of jobs (min)*		
	Gratis	Normal Production	Other
mean	186	6.79d	102
std	1653	10d	938
25%	10	170	15
50%	25	31.7h	25
75%	65	6.98d	55
90%	185	29d	120
95%	365	29d	255
99%	30.1h	29d	15.5h

Table 3: Distribution of job durations in the main priority classes. 29 days is the whole duration of the trace.

5.2. CPU usage variation

When considering allocation, it is important to categorize the correlation among jobs to be scheduled and/or that have already been scheduled. Indeed, if a resource provider knows that there is a high probability of having two jobs positively correlated in, say, CPU demand, it will take care to allocate them on different machines to avoid starvation of any of them in the event of a spike of demand. On the other hand, two negatively correlated jobs, allocated together, will allow for a better average utilization of the available resources. To perform this analysis, we have restricted to jobs in the *Normal Production* class, because jobs in the other priority classes do not last long enough, and thus analyzing correlations and usage patterns does not make sense. Furthermore, considering only one priority class avoids (at least partially) the correlation due to the fact that the platform has finite capacity. Indeed, this finite capacity implies that when the demand of one job increases, it uses more resources, and then another job may end up using less resources (or even getting evicted) even if its actual demand did not change.

Looking at individual resource usage of *Normal Production* jobs provided an interesting insight in their behavior: all jobs that are long enough present daily and sometimes weekly patterns. Since any signal is the sum of its Fourier components, it is possible to recreate the pattern of a group of jobs by just providing statistical values about the amplitude ratios between the main components, together with a small number of parameters such as frequency, amplitude and phase of the main components of each job, as well as a given amount of random noise.

We have analyzed the *Normal Production*, dominant jobs that run during the whole trace. The CPU demand of each job was decomposed into its Fourier series to quantify its main spectral components. After removing the harmonics, we quantified their amplitude, phase, frequency and background noise. Table 4 provides the averaged ratios between signal amplitudes and the constant part. We can see that the residual noise is about 6% of the average CPU demand for a large part of the jobs, and this can be used as a threshold: any pattern with an amplitude significantly larger can be identified as a relevant component. Our

Stats	*Ratio of amplitude to mean*				
	Hourly	*Daily*	*Weekly*	*Long term*	*Noise*
mean	0.057	0.267	0.148	0.154	0.100
std	0.246	0.232	0.127	0.161	0.154
min	0.001	0.006	0.011	0.001	0.012
25%	0.004	0.052	0.076	0.051	0.036
50%	0.007	0.268	0.106	0.102	0.058
75%	0.009	0.376	0.196	0.196	0.072
max	1.612	1.075	0.669	1.149	0.836

Table 4: Ratios Amplitude/DC for long-running, dominant jobs.

first observation is that very few jobs exhibit hourly patterns, whereas more than half of the jobs exhibit very strong daily patterns, and two thirds of the jobs have significant daily patterns. Weekly patterns are not as strong, but they are still significant for about half of the jobs. The long-term part of the signal represents variations on a larger time-scale, and we can see that this variation is also significant for many jobs, but it cannot be analyzed any further because the duration of the trace is only one month.

Another interesting question is how much these patterns are synchronized: if all jobs reach their peak demand at the same time, the stress on the platform and on the resource allocation algorithm is much higher. On the other hand, if the peaks are spread on a large enough timeframe, this provides some slack to the allocation algorithm to provide efficient allocations by co-allocating jobs whose peaks happen at different times. From our observations, jobs which exhibit a weekly pattern have all the same (synchronized) behavior: 5 days of high usage, followed by 2 days of lower usage. About daily patterns, we have analyzed the repartition of the phase for those jobs which exhibit a daily pattern (with an amplitude of at least 10% of the mean).

It appears that for half of the jobs, the phase difference is below 60 degrees, which means that their peaks are within 4 hours of each other. Furthermore, 90% of the jobs exhibit a phase difference below 120 degrees, which means that the peaks are at most 8 hours apart. This shows that the behavior of jobs are clearly correlated by this daily pattern, however there are indeed opportunities for good allocation strategies to make use of this 8 hour difference between the peak times of some jobs.

6. Machine failure characterization

The Google trace also includes information about machine events, in particular about times when some machines are removed from the system. According to the trace description, these "removed" events can be either machine crashes, or planned maintenance. An usual assumption for modeling fault-tolerance issues is that machines fail *independently*: the failure of one machine does not change the failure probability of other machines. Another assumption is that the failure probability of machines does not change with time. When both assumptions

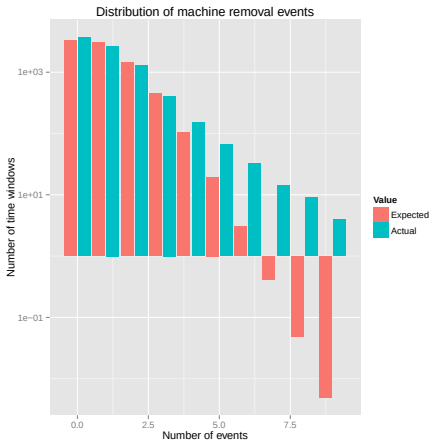


Figure 6: Actual versus expected distribution of failures

hold, the number of failures in a given time period is a random variable which follows a Poisson distribution $P(\lambda)$, where λ is the average number of failures.

In order to test these assumptions, we have counted the number of removals in all 8351 5-minutes time windows of the trace. The result gives an average number of removals of 1.07. Since the trace contains information about 12,000 machines, this would represent an (individual) machine failure probability of 0.8×10^{-4} in each time window. However, by looking closer into the data, we have identified three outstanding events, where the removal pattern is very different from the rest of the trace: two events where a large number of machines are removed at 30 seconds interval, and are all reinserted within a few seconds, and another event where a subset of the machines are removed and reinserted many times in a row, until all of them are eventually reinserted within a few minutes.

# events	0	1	2	3	4	5	6	7	8	9	10	11
# TW	3674	2679	1276	410	150	66	32	14	9	4	5	5

# events	13	14	15	16	17	18	19	20	23	25	30
# TW	2	3	7	1	2	3	4	1	1	3	1

By removing these three outstanding events, we obtain the table above, which shows how many 5-minutes time windows feature a given number of machine removals. The average number of removals is now 1.005. This table shows that in most time windows, the number of removals is rather small, but we can identify 38 time windows with at least 10 removals. Such occurrences are very unlikely under the Poisson distribution: the expected number of such time windows is less than 2.10^{-3}, which clearly indicates that some correlation exists. Of course, planned maintenance events are expected to be correlated, so the existence of such events could easily justify these occurrences, while keeping the independence assumption for "natural" failures.

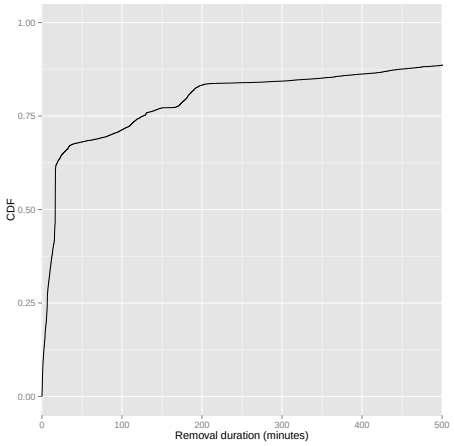


Figure 7: Failure durations distribution

For the next step, let us ignore the time windows that we expect to correspond to maintenance sessions, by assuming that all time windows with at least 10 removals are maintenance events. There are 8314 remaining time windows, for which the average number of events is 0.94. Figure 6 shows the actual number of time windows for each number of events, together with the expected value (under a Poisson distribution). This figure shows that time windows with more than 5 failures are still much more present in the trace than what would be expected from a Poisson distribution. This hints at the presence of some correlation on machine failure events, which should be taken into account in a complete model for machine failures. However, a more in-depth analysis would require to be able to differentiate between maintenance and actual failure events and as a first approximation, independent failures with a failure rate of 0.8×10^{-4} in each 5 minutes period can be considered as a reasonable model (this corresponds to a mean time between failures of about 1000 hours).

Another interesting feature about failures in this trace is that most removal periods are very short: in 60% of the cases, a removed machine is back in the system less than 17 minutes later. The complete CDF of removal durations is shown on Figure 7.

7. Modeling the workload

The work presented in previous sections quantifies the type of workload existing in actual datacenters such as Google's. From this analysis, we provide a set of parameters that allows for creating a reasonable, yet simple, model which can be used as an input of a system (for example, an algorithm) in the context of resource allocation.

There are several decisions that need to be taken during the model generation process. Among them, the number of degrees of freedom allowed. Simply put,

which parameters should we use in our model, how many of them, and how much can they vary? Too few parameters will result in a simplistic model that may not reliably reflect the features of the modeled workload. Too many parameters can overfit the model, so it will be more complex without this ensuring at all that it will provide better results. In fact, there is no such thing as a good or bad model, since it depends on the system to which it will be applied. For example, a system whose output is a function of the input memory usage will probably not care about job priorities or CPU usage. However, it might take into account, to a larger or lesser extent, the memory usage distribution and memory periodic patterns.

We believe that for such large datacenters with many tasks, the focus of the allocation system or algorithm should be on jobs themselves, which require large quantity of resources and need to be allocated on many machines, rather than on individual tasks. In fact, in many cases (*i.e.*, for many jobs) it might be possible for the allocation algorithm to decide on how the load of a job is balanced between its tasks. Even though this property cannot be inferred from the trace, it is hinted by the fact that for most jobs, there is a large variability of the CPU usage of its individual tasks. This assumption has been made for example in [19]. Another possibility is to consider a two-stage algorithmic process, in which the job allocation is computed globally, and then simple greedy allocation procedures are used to allocate individual tasks. In both cases, it is reasonable to consider that the memory usage of individual tasks is the same for all the tasks of a job. For more genericity, a linear dependency between CPU and memory usage for all tasks of each job may also be assumed.

In both cases, jobs should be described with their aggregate amount of CPU and memory usage. Our analysis of the dynamics of the trace has shown that it is crucial to model the variation of this resource usage, but that most of the correlation can be captured by considering daily variation patterns, or with more precision by considering both daily and weekly patterns. If required, the rest of the dynamics of job resource usage can be modeled as random noise, without dependencies between jobs.

Finally, machines can be assumed to have independent failures and have a failure rate of about 10^{-5} per hour.

8. Algorithmic studies

In this section, we present examples of algorithmic models derived from this trace analysis. These models are an attempt to identify meaningful allocation problems for which good algorithms or heuristics would increase the behavior and performance of Cloud systems.

8.1. Robust allocation

As a first model, we have studied a robust allocation problem [20], in which the objective is to allocate the jobs to machines so as to guarantee that each job achieves a prescribed reliability threshold. The assumptions and notations are the following:

- Each job is described by its computational and memory requirement, d_i and m_i, and an acceptable probability of failure r_i;
- Machines have homogeneous computational and memory capacities, C and M;
- We look for a static allocation, valid for a given period of time during which the allocation does not change;
- In an allocation, jobs can be divided in tasks and allocated to several machines. Each task can be allocated a different computational power, and we denote by $a_{i,j}$ the computing power allocated to job i on machine j, but all tasks require the total memory requirement of the job (in Section 4.2, we show that for most jobs, memory usage does not depend on the task CPU usage);
- During this period, some machines may fail with probability p, and we define X_j the random variable which is equal to 1 if machine j is alive at the end of the time period, and 0 otherwise. The allocation may thus provide more computational power to each job than required, so that the remaining computation power after failures occur $R_i = \sum_{j=1}^{m} X_j \times a_{i,j}$ is above the demand d_i with high probability.

The optimization problem is thus formalized as follows:

$$\text{Minimize } m \text{ such that } \begin{cases} \forall i, \mathbb{P}\left(R_i < d_i\right) < r_i & (1) \\ \forall j, \sum_{i=1}^{ns} m_i \mathbb{1}_{a_{i,j}>0} \leq M & (2) \\ \forall j, \sum_{i=1}^{ns} a_{i,j} \leq C & (3) \end{cases}$$

Thanks to this formulation based on jobs rather than tasks, we have been able to propose an efficient allocation algorithm [20], by using relaxation, reformulation and column generation techniques [21]. Using data from the dominant jobs in the Google trace, simulations have shown that this algorithm can provide in reasonable time very compact allocations.

8.2. Packing periodic jobs

As described in Section 5, another interesting feature of the jobs in this Google trace is the periodicity of resource demands. From a resource allocation perspective, it could be interesting to make use of this feature, by attempting to allocate on the same machine jobs which complement each other by having their peak demand at different times. Indeed, in standard allocation procedures, either jobs are provisioned their maximum resource demand to ensure that there is no overprovisioning, or dynamic strategies are used to migrate some of the tasks when overloading occurs. On the contrary, using the periodic feature of

jobs would allow to optimize resource usage with a limited use of migration. In this section, we propose one formulation for this problem, with the following notations and assumptions:

- Each job has a constant memory requirement m_i and is divided in a specified number of tasks n_i (in Section 4.3, we show that for most jobs, the number of tasks is constant over time);
- The workload of each task of job i is modeled as the sum of a constant part and a periodic part: $w_i(t) = c_i + a_i \sin(\phi_i + \omega t)$, where ω is chosen to have a periodicity over one day (in Section 5, we show that for most jobs, the daily variation pattern explains a large part of the variability, and in Section 4.3, we show that for many jobs, the total workload is evenly balanced between all tasks);
- Machines have homogeneous computational and memory capacities, C and M;
- Several tasks can be allocated to a machine if the total workload remains below the machine capacity at all time.

For a given set of tasks T, the total workload can be computed as $w_T(t) = \sum_{i \in T} c_i + \sum_{i \in T} a_i \sin(\phi_i + \omega t)$. Standard trigonometric computation shows that $w_T(t) = \sum_{i \in T} c_i + a_T \sin(\phi_T + \omega t)$, where a_T and ϕ_T are such that (with complex number notations) $a_T e^{\mathrm{i}\phi_T} = \sum_{i \in T} a_i e^{\mathrm{i}\phi_i}$. We can thus view the numbers (a_i, ϕ_i) as vectors $\overrightarrow{v_i}$ expressed with polar coordinates, and use them to express the resource constraint. Indeed, $w_T(t) \leq C$ for all t if and only if $\sum_{i \in T} c_i + a_T \leq C$, and a_T is simply the norm of the sum of the vectors $\overrightarrow{v_i}$, $a_T = \| \sum_{i \in T} \overrightarrow{v_i} \|$.

The corresponding optimization problem can thus be formalized as follows, where the variable $x_{i,j}$ denotes the number of tasks from job i allocated to machine j:

$$\text{Minimize } m \text{ such that } \begin{cases} \forall i, \sum_{j=1}^{m} x_{i,j} \geq n_i & (4) \\ \forall j, \sum_{i=1}^{ns} m_i x_{i,j} \leq M & (5) \\ \forall j, \sum_{i=1}^{ns} c_i x_{i,j} + \| \sum_{i=1}^{ns} x_{i,j} \overrightarrow{v_i} \| \leq C & (6) \end{cases}$$

This Second Order Cone Program [22] formulation can be solved with standard solvers like CPLEX, but real size instances are too large to obtain good solutions in a reasonable time. However, this mathematical formulation is a good step towards the design of efficient heuristics for this optimization problem.

9. Conclusions and future work

We have provided in this paper a detailed analysis of a Google Cluster usage trace. We fulfilled the objective of proving that, even if the general resource allocation problem is amenable to a very difficult multi-dimensional bin packing problem, a number of specific features of the trace makes it possible to model jobs with very few parameters, which allows for the design of efficient resource allocation algorithms for Clouds.

Most of the previous articles in the literature highlight the dynamics and heterogeneity of the trace. However, we have shown that there are a set of common features that allow for a simplification of the workload while maintaining the essential behavior needed for the design of trace-generating models. To this end, we focused on the so-called dominant jobs, which account for 90% of the CPU usage of the platform. This set of jobs is relatively smaller, which eases the design of sophisticated allocation techniques.

We carried out an exhaustive study on this set of dominant jobs. The considered features included the resource usage distribution (statically and dynamically), job priorities, variability in task number and resource consumption, periodicity and scale of the existing patterns, independence among resource dimensions (CPU and memory), and job lifetime and failure distributions.

It is noticeable that, in contrast to the conclusions drawn in [7] which point out a big difference between traces from Google and from Grids, we found out that the dominant jobs show a closer resemblance and might benefit from techniques and models used for Grid workloads. For example, despite the higher variability in the number of tasks per job and the higher submission frequency, the load balance variation of the dominant jobs is moderate in general (as opposed to the whole trace), which is closer to a Grid's behavior. In addition, dominant jobs are longer than the average duration of Grid's jobs, in contrast to the rest of jobs of the trace. Finally, whereas the CPU usage in the Google cluster is usually lower than the memory usage, in the case of the dominant jobs the CPU usage is slightly higher than the memory usage, closer to the usage of typical Grid jobs.

This work opens a number of interesting questions. On the trace analysis side, we have identified some parameters of interest, and it would be very valuable to propose a complete generating model of those parameters. The characterization of machine failures over time would also be very interesting, but such a detailed analysis would require to differentiate between failures and maintenance. On the algorithmic side, we plan to use the framework we have proposed in this paper to design and validate efficient resource allocation algorithms to cope with (and to benefit from) the dynamics of resource usage.

Acknowledgments

This material is based upon work supported by the French National Research Agency (ANR) under contract no. ANR-11-INFRA-13.

References

[1] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, M. A. Kozuch, Heterogeneity and dynamicity of clouds at scale: Google trace analysis, in: ACM Symposium on Cloud Computing (SoCC), San Jose, CA, USA, 2012.

[2] J. Wilkes, More Google cluster data, Google research blog, posted at `http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html`. (Nov. 2011).

[3] C. Reiss, J. Wilkes, J. L. Hellerstein, Google cluster-usage traces: format + schema, Technical report, Google Inc., Mountain View, CA, USA, revised 2012.03.20. Posted at URL `http://code.google.com/p/googleclusterdata/wiki/TraceVersion2` (Nov. 2011).

[4] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, M. A. Kozuch, Towards understanding heterogeneous clouds at scale: Google trace analysis, Tech. rep., Carnegie Mellon University (Apr. 2012).

[5] Google cluster data discussion group, `https://groups.google.com/forum/#!forum/googleclusterdata-discuss`.

[6] S. Di, D. Kondo, F. Cappello, Characterizing cloud applications on a Google data center, in: 42nd International Conference on Parallel Processing (ICPP2013), Lyon, France, 2013.

[7] S. Di, D. Kondo, W. Cirne, Characterization and comparison of cloud versus Grid workloads, in: International Conference on Cluster Computing (IEEE CLUSTER), Beijing, China, 2012, pp. 230–238. `doi:10.1109/CLUSTER.2012.35`.

[8] Z. Liu, S. Cho, Characterizing machines and workloads on a Google cluster, in: 8th International Workshop on Scheduling and Resource Management for Parallel and Distributed Systems (SRMPDS), Pittsburgh, PA, USA, 2012.

[9] P. Garraghan, P. Townend, J. Xu, An analysis of the server characteristics and resource utilization in google cloud, in: Proceedings of the 2013 IEEE International Conference on Cloud Engineering, IC2E '13, IEEE Computer Society, Washington, DC, USA, 2013, pp. 124–131. `doi:10.1109/IC2E.2013.40`.

[10] S. Di, Y. Robert, F. Vivien, D. Kondo, C.-L. Wang, F. Cappello, Optimization of cloud task processing with checkpoint-restart mechanism, in: 25th International Conference on High Performance Computing, Networking, Storage and Analysis (SC), Denver, CO, USA, 2013.

[11] S. Di, D. Kondo, W. Cirne, Host load prediction in a Google compute cloud with a Bayesian model, in: International Conference on High Performance Computing, Networking, Storage and Analysis (SC), IEEE Computer Society Press, Salt Lake City, UT, USA, 2012, pp. 21:1–21:11.

[12] M. Amoretti, A. Lafuente, S. Sebastio, A cooperative approach for distributed task execution in autonomic clouds, in: 21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), IEEE, Belfast, UK, 2013, pp. 274–281. doi:10.1109/PDP.2013.47.

[13] A. Ali-Eldin, M. Kihl, J. Tordsson, E. Elmroth, Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control, in: 3rd Workshop on Scientific Cloud Computing (ScienceCloud), ACM, Delft, The Nederlands, 2012, pp. 31–40. doi:10.1145/2287036.2287044.

[14] N. Bobroff, A. Kochut, K. Beaty, Dynamic placement of virtual machines for managing sla violations, in: Integrated Network Management, 2007. IM '07. 10th IFIP/IEEE International Symposium on, 2007, pp. 119–128. doi:10.1109/INM.2007.374776.

[15] D. Gmach, J. Rolia, L. Cherkasova, A. Kemper, Workload analysis and demand prediction of enterprise data center applications, in: Workload Characterization, 2007. IISWC 2007. IEEE 10th International Symposium on, 2007, pp. 171–180. doi:10.1109/IISWC.2007.4362193.

[16] M. R. Garey, D. S. Johnson, Computers and Intractability, a Guide to the Theory of NP-Completeness, W. H. Freeman and Company, 1979.

[17] D. Hochbaum, Approximation Algorithms for NP-hard Problems, PWS Publishing Company, 1997.

[18] M. Ester, H. peter Kriegel, J. S, X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, AAAI Press, 1996, pp. 226–231.

[19] O. Beaumont, P. Duchon, P. Renaud-Goud, Approximation Algorithms for Energy Minimization in Cloud Service Allocation under Reliability Constraints, in: HIPC, HIgh Performance Computing, Bengalore, Inde, 2013. URL http://hal.inria.fr/hal-00788964

[20] O. Beaumont, L. Eyraud-Dubois, J.-A. Lorenzo, P. Renaud-Goud, Efficient and robust allocation algorithms in clouds under memory constraints, in: Proc. of the 21st IEEE Conference on High Performance Computing (HIPC), 2014.

[21] J. Desrosiers, M. E. Lübbecke, A primer in column generation, Springer, 2005.

[22] S. Boyd, L. Vandenberghe, Convex optimization, Cambridge university press, 2009.